土木建筑工程信息技术
Journal of Information Technology in Civil Engineering and Architecture

基于 HBase 的 BIM 模型存储技术研究

陈　远　岳石花

（郑州大学土木工程学院，郑州　450001）

【摘　要】随着计算机互联网技术的飞速发展，建筑信息模型技术（BIM）应运而生。BIM 作为一种新兴的建筑技术理念，推动着建筑行业发生巨大的变革，同时，大数据、云计算技术也成为当下热门话题。将大数据云计算与 BIM 技术相结合，将是未来建筑行业的发展趋势。本文研究如何将 BIM 模型存储在基于大数据平台的开源 HBase 数据库中。通过对 BIM 模型的开放性标准 IFC 的内部结构进行研究及解析，设计出基于 IFC 的 HBase 数据库的格式，然后开发基于 IFC 格式的 HBase JAVA API 总体类库。最后通过运用 HBaseJAVA API 存储示例说明存储的技术方法。

【关键词】BIM；IFC；HBase；JAVA API；数据存储
【中图分类号】TP311.13　【文献标识码】A　【文章编号】1674－7461（2017）04－0074－08
【DOI】10.16670/j.cnki.cn11－5823/tu.2017.04.14

前言

传统建筑行业数据量多，数据收集困难，数据交换效率低下。建筑信息模型（BIM）是以三维表达技术为基础，集成了工程项目的各种信息，其核心技术是要实现建设项目各个参与方的信息共享与交换，解决以往建筑信息孤岛的问题。IFC 标准作为开放的 BIM 标准，是一个计算机可以处理的建筑数据表示和交换标准，其目的是支持工程项目全寿命周期内各个阶段信息的共享与交换[1]。IFC 标准是实现不同应用软件之间数据交流与互操作的中间件，它不依赖任何一个具体的信息系统，也不受任何一个或一类软件开发商的控制，并且能够用于描述建筑项目全生命期内不同专业的所有产品数据信息[2]。然而以往的基于 IFC 文件式的交换方式已经无法满足日益复杂的工程的需求。BIM 数据存储技术是建筑行业需要解决的首要问题。

随着互联网、云计算技术的迅速发展，大数据在医疗、卫生、教育、工业行业等行业已经取得了巨大的经济效益，采用大数据技术来辅助现代建筑行业也必将为建筑业带来巨大的影响。将建筑行业与大数据技术相结合，首先要解决的就是如何将建筑数据存储到大数据平台，进而进行建筑数据的管理和分析。HBase 数据库作为开源的大数据平台 Hadoop 的一部分，解决了海量数据的高效存储问题。因此，本文将探讨通过 HBase JavaAPI 将基于 IFC 格式的 BIM 模型储存到 HBase 数据库的具体技术和方法。

1　IFC 标准的概述

1.1　IFC 整体框架

IFC 标准是由国际协作联盟 IAI（the International Alliance for Interope-rability，现名为 BuildingSMART）提出，用于解决不同专业之间以及同一专业之间信息交换与共享。IFC 标准自底层到上层依次包括：资源层、核心层、共享层和领域层。资源层主要是产品基础信息的定义，例如日期、人员、几何信息、成本信息、材料资源等等。核心层通过引用资源层信息来描述模型的整体框架，然后来描述真实世界中存在的实物及抽象的一些流程。共享层用来解决跨专业领域的信息交互问题，同时细化各系统的组成元素，如门、窗、墙、梁、柱等常用的建筑实体都在该层被定义。领域层是 IFC 模型体系的最高层，该层主要定义了每个具体领域（建筑领域、结构领

【作者简介】　陈远（1975－），男，博士，副教授。主要研究方向：土木建筑工程信息化。

域、设备管理领域等)的实体概念。在 IFC 框架中领域层、共享层、核心层所有的实体都有一个共同的抽象基类 IfcRoot,且都具有唯一的标识(GUID)。资源层的实体不能独立存在通常作为上层中实体的属性存在。

1.2 基于 IFC 的实体表达

完整的 IFC 文件模型由类型定义、计算实体属性值的函数和规则、以及常用属性集的预定义组成。类型定义又包括实体类型、选择类型、定义类型和枚举类型四种。其中实体类型是类型定义的核心也是信息交换与共享的载体,其等同于面向对象程序语言中类的概念。类型实体中其他类型的功能主要是作为属性值被实体实例所引用。

在 IFC 中性文件中,任何实体都是通过属性来表达描述自身的信息。这些属性可以分为直接属性、导出属性和反转属性。如图 1。在 IfcWall 实体中直接属性是一些直接信息或指标量如 GlobalId、Name 等。导出属性是指由其他实体来表达的属性,如 OwnerHistory、ObjectPlacement、Representation 等。反转属性是关键字“INVERSE”之后定义的一些属性,通过关联实体来定义实体实例之间“双向的”一对多或多对多的关联关系。

1.3 IFC 实体分类

IFC 信息交换的载体是实体,实体是 IFC 信息交换的最小单元。IFC 中的实体按是否可以独立交换划分为可独立交换实体和不可独立交换实体。如表 1 所示,可独立交换实体都是继承自 IfcRoot。将 IfcRoot 的派生实体 IfcObject、IfcRelationship 以及 IfcPropertyDefinition 按照在信息交换中所起到的作用来划分,又可划分为主体实体和辅助实体,如表 2 所示。

表 1 IFC 实体的分类

分类方式	类别	说明
按照是否可独立交换	可独立交换	可独立交换的实体继承自 IfcRoot,具有唯一的 ID(GUID)具备全局标识特性分布在核心层、交互层和领域层。
	不可独立交换(资源层实体)	不可独立交换的实体分布在资源层,又称为资源实体。不是继承自 IfcRoot,没有 GUID,不具备全局标识特性,通常作为可独立交换实体的属性存在。

表 2 IfcRoot 派生实体的分类

分类方式	类别	说明
按照在信息交换的作用分类	主体实体	主体实体是指由对象实体(IfcObject)所派生出来的实体。包括 IfcProduct(产品)、IfcProcess(过程)、IfcControl(控制)、IfcResource(资源)、IfcActor(角色)、IfcProject(项目)和 IfcGroup(分组)及其子类。这些实体在信息交换中起主导作用。
	辅助实体	辅助实体是指在 IfcRoot 中除去主体实体(对象实体)以外的实体,包括 IfcRelationship、IfcPropertyDefinition 及其子类,这些辅助实体对主体实体起到修饰、扩充定义关系等作用。

1.4 IFC 文件的解析

IFC 标准是基于 EXPRESS 语言来描述建筑数


```
ENTITY IfcWall;
  ENTITY IfcRoot;
    GlobalId                    : IfcGloballyUniqueId;
    OwnerHistory                : IfcOwnerHistory;
    Name                        : OPTIONAL IfcLabel;
    Description                 : OPTIONAL IfcText;
  ENTITY IfcObjectDefinition;
  INVERSE
    HasAssignments              : SET OF IfcRelAssigns FOR RelatedObjects;
    IsDecomposedBy              : SET OF IfcRelDecomposes FOR RelatingObject;
    Decomposes                  : SET [0:1] OF IfcRelDecomposes FOR RelatedObjects;
    HasAssociations             : SET OF IfcRelAssociates FOR RelatedObjects;
  ENTITY IfcObject;
    ObjectType                  : OPTIONAL IfcLabel;
  INVERSE
    IsDefinedBy                 : SET OF IfcRelDefines FOR RelatedObjects;
  ENTITY IfcProduct;
    ObjectPlacement             : OPTIONAL IfcObjectPlacement;
    Representation              : OPTIONAL IfcProductRepresentation;
  INVERSE
    ReferencedBy                : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
  ENTITY IfcElement;
    Tag                         : OPTIONAL IfcIdentifier;
  INVERSE
    HasStructuralMember         : SET OF IfcRelConnectsStructuralElement FOR RelatingElement;
    FillsVoids                  : SET [0:1] OF IfcRelFillsElement FOR RelatedBuildingElement;
    ConnectedTo                 : SET OF IfcRelConnectsElements FOR RelatingElement;
    HasCoverings                : SET OF IfcRelCoversBldgElements FOR RelatingBuildingElement;
    HasProjections              : SET OF IfcRelProjectsElement FOR RelatingElement;
    ReferencedInStructures      : SET OF IfcRelReferencedInSpatialStructure FOR RelatedElements;
    HasPorts                    : SET OF IfcRelConnectsPortToElement FOR RelatedElement;
    HasOpenings                 : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
    IsConnectionRealization     : SET OF IfcRelConnectsWithRealizingElements FOR RealizingElements;
    ProvidesBoundaries          : SET OF IfcRelSpaceBoundary FOR RelatedBuildingElement;
    ConnectedFrom               : SET OF IfcRelConnectsElements FOR RelatedElement;
    ContainedInStructure        : SET [0:1] OF IfcRelContainedInSpatialStructure FOR RelatedElements;
  ENTITY IfcBuildingElement;
  ENTITY IfcWall;
END_ENTITY;
```

图 1 IfcWall 属性继承

据。EXPRESS 是一种面向对象的信息描述性语言，不是编程语言，不能直接用于编程开发。因此需要对 IFC 模型进行解析，然后才能运用计算机语言如 C#、C + + 、JAVA 等来通过编程具体实现所需的功能。

目前用于 IFC 解析的软件有很多，Jotne EPM Technology 公司的 EDM；基于 C + + 的 IfcPlusPlus；基于. NET 平台的 xBIMToolKit；基于 JAVA 的 Ifc-ToolsProjcet 等。本文采用的是基于 Java 的 IfcToolsProjcet 中的 IFC JAVA Toolbox 工具。该工具可以方便的阅读、编写、修改和添加 IFC 文件。IFC JAVA Toolbox 支持 IFC 模式中所有实体的直接属性和反转属性。针对 IFC 中的每个实体，IFC JAVA Toolbox 中都有一个对应的 Java 类，其核心模型 IfcModel 为提取和设置 IFC 对象的任何显式属性和反转属性提供方法，因此可以实例化并处理任何对象[3]。创建一个 IfcModel 类作为获取工程的入口类，该类包含了模型中所有的实体。通过 ifcModel. readStepFile ( )对 IFC 模型的信息进行读取与解析加载。具体的实现方式如图 2 所示。

每个 IFC 文件有且仅有一个 IfcProject 对象，它是整个 IFC 文件的最高级别的实体类型，所以无论是利用 IFC 文件临时传递一个零部件、一根杆件，还是装载整个建筑工程，都应该从 IfcProject 开始，自上而下地逐层定义对象，直到完整表达所要表达的模型信息[4]。通过分析 IFC 模型文件，先获得该模型所定义的 IfcProject 对象、IfcSite 对象 IfcBuilding

```
//定义IfcModel类
private IfcModel createIfcModel() {
  // 创建一个IfcModel实例
IfcModel ifcModel = new IfcModel();
      // 从文件系统中加载 IFC STEP文件
File stepFile = new File("f:\\ifcpractise \\某教学楼项目.ifc");
try {
            ifcModel.readStepFile(stepFile);
      } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
      return ifcModel;
}
```

图 2　IFC 文件的获取与解析

对象、IfcBuildingStorey 对象等以及与所有建筑楼层对象关联的墙(IfcWall) 、梁(Ifc Beam) 、柱(IfcColumn) 、板(IfcSlab) 、楼梯(IfcStair) 、窗(IfcWindow) ，门(IfcDoor) 等结构构件对象，这些结构构件对象是主体建筑的主要构件，也是工程数据交换的核心[5]。对于获取到的对象实体通过 IFC JAVA Toolbox 中的 get-methods 和 set-methods 来读取和设置特定的 IFC 实体的属性。对于 IFC 实体反转属性可通过各自的 get 方法获得，不需要 set 方法，因为他们是自动解决的。

2　HBase 数据库的概述

2. 1　HBase 数据库整体架构

NoSQL(非关系型数据库) 是一种新的数据管理专门设计的技术来满足要求的数据不同，它不需要一个预定义的数据模式，这意味着它不仅使 IFC 架

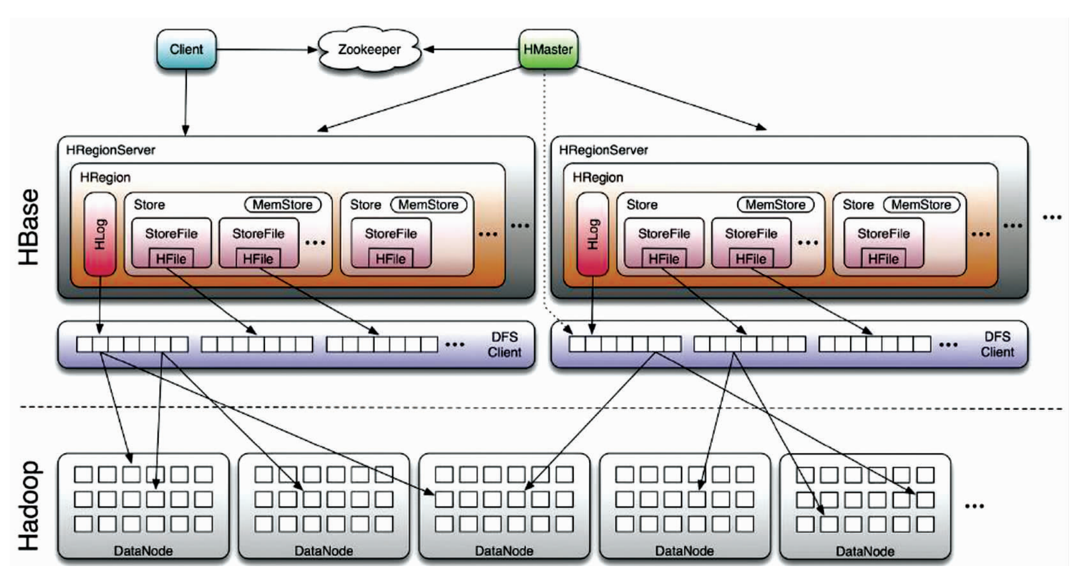


图 3　HBase 系统架构

构中的所有数据定义映射到数据库结构中,而且也可以添加新功能的数据结构实体,这符合 IFC 的发展模式,不断采用新的建模概念到新的版本中[6]。

HBase(Hadoop Database)作为一种 NoSQL 数据库是基于 GoogleBigTable 的开源实现,主要用于存储海量的数据。它是构建在 Hadoop 中的 HDFS 之上的分布式的面向列的开源数据库。利用 Hadoop 中的 MapReduce 来处理数据。利用 Zookeeper 来处理协同服务。HBase 的系统架构图如图 3 所示。

客户端 Client 是整个集群的入口,用户可以直接通过 Client 操作 HBase,使用 HBase RPC(远程调用)机制与 HMaster 和 HRegionServe 通信。Client 与 HMaster 通信是进行管理类的操作,与 RegionServe 通信是进行数据读写类操作。

程序的协调服务 Zookeeper 是集群中的协调者,其主要功能有:保证了集群中任何时候只有一个 Master;实时监控 RegionServe 的上线和下线信息,并通知给 Master;存储所有 Region 的寻址入口;存储 HBase 的 schema 和 table 元数据。

HBase 中主节点 HMaster 管理用户对 Table 的增删改查,管理 HRegionServe 的负载均衡,调整 Region 分布。在 Region Split 后,负责新 Region 的分配。

HRegionServe 维护 Region 并往 HDFS 中书写数据,是 HBase 中最核心的模块。管理 HBase 中逻辑表的结构,当表的大小超过设计值时,Split HRegion。

2.2 HBase 数据库存储模式

从逻辑上看 HBase 以表的形式存储数据,表由行和列组成,每个列是属于某个列族。在 HBase 中并不是每行每列都存储数据,它是一个稀疏的表。在 HBase 中也不存在 null(空记录)。当存在没有数据的情况时,HBase 不会存储任何东西。HBase 的逻辑模型如表 3 所示。关于表中的几个概念的介绍如下:

(1)行键(Rowkey)

Rowkey 是数据行在表里的唯一标识,作为检索记录的主键。Rowkey 可以为最大长度不超过 64KB 的任意字符串,并按照字典序存储。访问 HBase 表中的行有三种方式:

· 通过单个行键(Rowkey)访问;

· 通过给定行键的范围访问;

· 全表扫描。

(2)列族(ColoumFamily)

在 HBase 表格中列(Coloum)是属于列族(ColoumFamily)的一部分,列族是列的集合,一个列族可以包含多个列,列族支持动态的扩展,在建表时不需要预先定义列的数量以及类型,列中的数据都是以二进制存储,没有数据类型。列可以表示为 <列族>:<限定符>;这就表明一个列族中所有的列成员都有相同的前缀;例如:列 courses:English,列 courses:Math,和列 courses:History 都是列族 courses 的成员,“:”是分隔符,用来区分前缀和列名。

(3)时间戳(Timestamp)与存储单元(Cell)

HBase 中单元格(Cell)由 Rowkey,ColoumFamily,Coloum,Timestamp 唯一确定,每一个存储单元都保存着同一份数据的多个版本,这些不同的版本通过时间戳来索引。在写入数据时 Timestamp 可以由 HBase 自动赋值也可以手动赋值,最终不同的版本按照时间倒序排序,即最新的数据显示在前面。一般系统默认保存 3 个版本。

表 3　HBase 逻辑模型

行键(Rowkey)	时间戳(Timestamp)	列族(ColoumFamily)	
		Coloum1	Coloum2
1	t3	Coloum1 Value3	Coloum2 Value3
	t2	Coloum1 Value2	Coloum2 Value2
2	t1	Coloum1 Value1	Coloum2 Value1
	t2	Coloum1 Value5	Coloum2 Value5
	t1	Coloum1 Value4	Coloum2 Value4

2.3 HBase JAVA API 简介

HBase API 由多种形式,但由于 HBase 的开发语言是 Java 语言,Java API 是最有效的访问方式。本文主要研究原生 Java API 形式。主要介绍五大类:HBaseConfiguration、HBaseAdmin、HTableDescriptor、HTable、HBase CRUD Operation。

(1)HBaseConfiguration

在使用原生 Java 客户端前,首先要配置客户端,HBaseConfiguration 是用于客户端的配置类,它是每一个 Hbase Client 都会使用到的对象。可以通过如下代码构建该对象:

Configurationconfig = HBaseConfiguration. create()

(2)HBaseAdmin

HBaseAdmin 类是 HBase 数据库的管理入口类，提供一个接口来管理 HBase 数据库的表信息，并管理表格的元数据。它提供的方法包括：创建表、删除表、修改表、使表有效或无效，以及添加或删除表列族成员等。

（3）HTableDescriptor

HTableDescriptor 主要是对表里的列族进行描述。可以通过如下代码构造该对象：

HTableDescriptort = new HTableDescriptor（tableName）

（4）HTable

在 HBase 中 HTable 用来封装表格对象，对表格的增删改查操作主要通过它来完成，当构建多个 HTable 对象时，HBase 推荐所有的 HTable 使用同一个 Configuration。这样，HTable 之间便可共享 HConnection 对象、zookeeper 信息以及 Region 地址的缓存信息。

（5）HBase CRUD Operation

HBase 的 CRUD 操作接口用到了在 hbase. client 包下的 org. apache. hadoop 中的 HTable 类，它提供了用户所需的所有的存储，从 HBase 中检索数据以及删除过时的值等功能[7]。对表的创建、删除、显示以及修改等，可以用 HBaseAdmin，一旦创建了表，那么可以通过 HTable 的实例来访问表，每次可以往表里增加数据。

1）put 方法

①单行 put 操作

向 HBase 中存储数据可以通过如下代码来实现该功能的调用：

Void put（Put put）throws IOException

这个方法以单个 Put 或存储在列表中的一组 Put 对象作为输入参数。创建 put 实例时，用户需要提供一个行键 row，在 HBase 中每行数据都有唯一的行键（rowkey）作为标识。创建 put 实例后，就可以向该实例添加数据。使用 add（）方法把数据添加到 put 实例中。如果再添加一个时间戳就能形成一个单元格。

②批量 put 操作

批量插入 put 实例的调用形式如下：

Void put（List <Put> puts）throws IOException

批量操作 put，用户需要创建一个列表 List 来存放所有的 Put 实例，将想要存放的 Put 实例依次放到列表 List 上来，最后再执行 puts 操作，把列表 List 中的 put 实例都更新存放到 HBase 上。

2）get 方法

从客户端 API 获取已存储数据，使用 Get 对象，通过 HTable. get（Get）来调用。HTable 提供了 get（）方法，同时还有与之对应的 Get 类。get 方法分为两类：一类是一次获取一行数据，另一类是一次获取多行数据。

①单行 get 操作

与 put 操作一样，在检索数据时，要按照一个指定的行键（rowkey）来创建用来 get 的实例。其调用形式如下：

Result get（Get get）throws IOException

②批量 get 操作

与 put 操作一样 get 操作也可以进行批量的操作。其操作流程和思路与 put 操作一样，不再赘述。其调用形式如下：

Result[ ]get（List <Get> gets）throwsIOException

3）delete 方法

①单行 delete

与前面的 get（）方法和 put（）方法一样，用户必须先创建一个 Delete 实例，然后添加想要删除的数据的详细信息。其调用形式如下：

Void delete（Delete delete）throws IOException

②批量 delete

基于列表 List 的 delete（）调用与基于列表的 put（）调用一样需要先创建一个包含 Delete 实例的列表，对其进行配置，并调用如下方法：

Void delete（List <Delete> deletes）throws IOException

3 基于 HBase 数据库的 IFC 信息存储的方法

3.1 基于 IFC 的 HBase 数据库设计

数据库的设计是 API 层设计和应用层设计的基础，只有将数据库的表格设计的符合逻辑、符合需求，才能设计合理的 API 层和应用层[8]。HBase 数据库接口是为了给上层更好的对其进行数据存取而根据 HBase 提供的 API 写的类包，其主要实现了两个部分的功能，一是对数据库表级的操作，二是对数据库中数据存取的操作[9]。

IFC 中的实体其属性可以分为直接属性、导出属性和反转属性。在 IFC 标准中直接属性以字符串形式来直接描述实体的相应属性。IFC 中的导出属

性对应的是一个 IFC 实体,将该实体实例的二进制序列化值存储在 HBaseTable 中。IFC 中反转属性可以通过关联相应的 IfcRelationship 表得到其属性。如表 4 所示。在 IfcProject 表中,IfcProject(项目)实体直接属性由 Name、Description、Object Type、Phase、LongName 这些属性在表中以字符串的形式直接描述对应的属性值。导出属性如 OwnerHistory 由 IfcOwerHistory 实体对象的二进制序列化值来得到其属性值。反转属性如 HasAssigenments 通过关系实 IfcRel Assigns 来描述该属性。要获得 HasAssigenments 属性需要先获取 IfcRelAssigns 及其子类的关系对象,再通过关系对象指向一个具体的相关实例对象。又如 HasAssociations 通过关联实体 IfcRelAssociates 可以关联构件的材料信息。

对于 IFC 中的可独立交换的实体建立相应的 HBase 表格,表中以直接属性、导出属性和反转属性作为列族。对于资源层中的资源实体不用建立相应的表格,其作为可独立交换实体的属性存在。在 HBase 数据库中将 IFC 可独立交换的实体中的所有主体实体建立一张大表。对于辅助实体即关系实体和类型实体是主体实体的属性的那些类型可以在主体实体表中仅保存其 GlobalId。

3.2 HBase JAVA API 的设计

为了使用户能直接对 IFC 数据进行编程扩展,需对 IFC 数据进行封装,进而提供统一的 API(Application Programming Interface,应用程序编程接口),API 设计的合理性是数据库扩展的关键[10]。本文设计的基于 IFC 的 HBase JAVA API 的对象模型主要包含三个基本类:项目类、主体实体类,关系实体类。如图 4 所示。

每一个 IFC 文件有且仅有一个项目类(IfcProject),该类位于信息交换的顶端。项目类(Project)主要存储项目的基本信息。包括项目的名称、项目的编号、项目所处的阶段。项目类所有的属性信息如表 4 所示。

API 中的主体实体类主要指工程中继承自 IfcProduct 的物理对象。该对象实体是所有数据存储的依据,也是工程中的主要内容。在 IFC 标准中建筑产品通过运用实体 IfcProduct 来定义,IfcProduct 表示一个对象描述的几何表示和局部位置[11]。该类中包含了项目的场地类、建筑类、楼层类、相关的建筑构件类如墙柱、梁、板、楼梯、屋顶等。

表 4 IfcProject 表

IfcProject(项目)		
RowKey		IfcProject 的 GlobalId 值
TimeStamp		时间戳
列族	列	说明
直接属性(Direct)	Name	名称(字符串)
	Description	描述(字符串)
	ObjectType	对象类型(字符串)
	Phase	工程所处阶段(字符串)
	LongName	长名(字符串)
导出属性(Derive)	OwnerHistory	"IfcOwerHistory"对象实例的二进制序列化值
	Representation Contexts	"IfcRepresentationContexts"对象实例的二进制序列化值
	UnitsInContext	"IfcUnitsAssigenment"对象实例的二进制序列化值
反转属性(Inverse)	HasAssigenments	通过一个分配关系 IfcRelAssigns 分配其他 IfcObject 子类型给这个对象
	IsDecomposedBy	通过关系实体 IfcRelDecomposes 来允许这个对象是由其他对象合成
	Deconposes	通过关系实体 IfcRelDecomposes 来允许这个对象是分解的一部分
	HasAssociations	通过关系实体 IfcRelAssociates 定义在 IFC 资源层概念的关系集。例如分类、库或文档引用
	IsDefinedBy	通过关系实体 IfcRelDefines 进一步定义对象的类型或属性信息的关系集(静态或动态的定义)

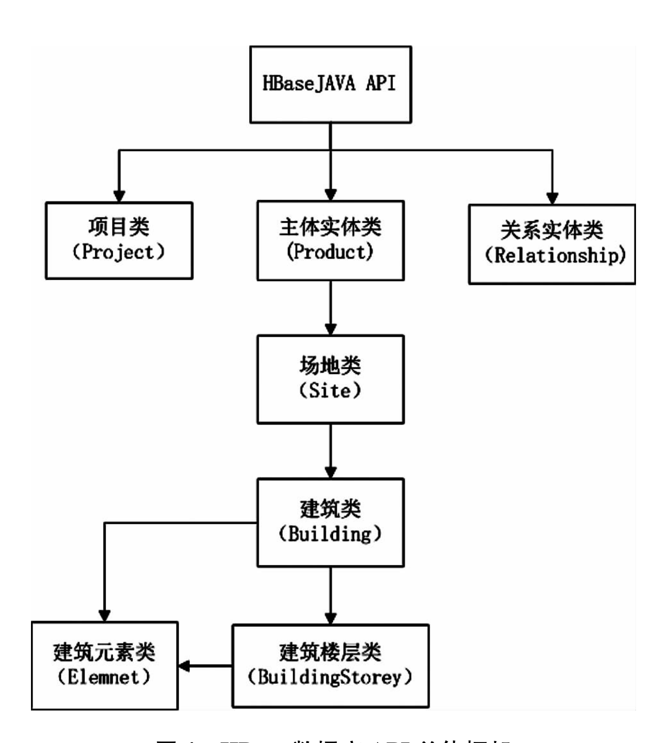


图 4 HBase 数据库 API 总体框架

在主体实体中通过关系实体类可以实现实体与实体、实体与属性之间的定义，对象化的关系类将实体引用保存在自身实例中，这些引用关系在实体中表现为反转属性。因此关系类是连接项目中实体与实体、实体与属性之间的关系的。该类的存在依附于项目类和主体实体类。

3.3 利用 HBase JAVA API 实现存储的示例

(1)连接 HBase 数据库

在 Java 代码中，为了连接到 HBase，我们首先创建一个配置(Configuration)对象，使用该对象创建一个 HTable 实例。这些配置文件信息包括 Zookeeper 地址、主机地址等。这个 HTable 对用于处理所有的客户端 API 调用。通过语句“Configuration conf = HBaseConfiguration. create( )”来获取 HBase 数据库中的配置信息，默认的构造方式会从 hbase-default. xml 和 hbase-site. xml 中读取配置，如果 classpath 中没有这两个文件，需自己配置。通过“connection = ConnectionFactory. createConnection(conf)”来建立连接。如图 5。

```
public class HBaseUtil {
  private static Configuration conf;
  private static Connection connection;
 // 初始化连接
  static {
    conf = HBaseConfiguration.create(); // 获得配制文件对象
    conf.set("hbase.zookeeper.quorum","192.168.52.140");//设置配置参数
  try {
      connection     =     ConnectionFactory.createConnection(conf);//
获得连接对象
      admin=connection.getAdmin();
   } catch (IOException e) {
      e.printStackTrace();
    }
 }
```

图 5 创建连接代码

(2)创建表

需要用 HTableDescriptor 构建一张表，并且使用 HColumnDescriptor 添加一个或多个列簇，然后调用 createTable 方法建表。完成之后再在表中添加一些数据。如图 6 所示。

(3)添加数据

对于解析后的 IFC 文件通过前面所述 getCollections 方法及 get-method 方法等可以获取到所有实体集合及其属性集合。用 Hbase 中 API 代码循环读取这些集合，将读取的到集合对应放入 HBase 表中。添加数据时创建 put 对象，利用 rowkey 添加一行数据，Put 的构造函数都需要指定行键。添加具体数据调用 put. add( ) 方法。所有的数据都被添加之后，我们调用 HTable. put( ) 方法会将数据保存进 HBase 的 table 中。也就是说创建 Put 对象及 put. add 过程都是在构建一行的数据，创建 Put 对象时相当于创建了行对象，add 的过程就是往目标行里添加 cell，直到 table. put 才将数据插入表格。部分代码如图 7 所示。

```
// 新建一个表的描述
HTableDescriptortableDesc=newHTableDescriptor(IfcProject);
// 在描述里添加列族
for (String columnFamily : columnFamilys) {
tableDesc.addFamily(newHColumnDescriptor("Direct"));
tableDesc.addFamily(newHColumnDescriptor("Derive "));
tableDesc.addFamily(newHColumnDescriptor("Inverse "));
}
// 根据配置好的描述建表
hAdmin.createTable(tableDesc);
System.out.println("创建表 "+ IfcProject +" 成功!");
```

图 6 创建表的部分代码

```
Put put=null;
//循环读取
for (String str : set) {
//指定添加行,设置RowKey,该行键为IfcProject的22为GUID值
Put put = new Put(Bytes.toBytes("0cs8HZC$1BhfpOzkrKVrbn "));
//往列族Driect中添加数据
put.add(Bytes.toBytes(Driect), Bytes.toBytes(Name),  Bytes.toBytes(value));
put.add(Bytes.toBytes(Driect), Bytes.toBytes(Description),  Bytes.toBytes(value));
..............
//往列族Derive中添加数据
put.add(Bytes.toBytes(Derive), Bytes.toBytes(OwerHistory),  Bytes.toBytes(value));
put.add(Bytes.toBytes(Derive), Bytes.toBytes(UnitsInContext),  Bytes.toBytes(value));
..............
//往列族Inverse中添加数据
put.add(Bytes.toBytes(Inverse), Bytes.toBytes(HasAssigenments),  Bytes.toBytes(value));
..............
}
 table.put(put);
 table.flushCommits();
 table.close();
```

图 7 添加数据部分代码

4 结论与展望

本文在大数据背景下研究将基于 IFC 标准的建筑信息模型(BIM)存储到 HBase 数据库中，并且能实现对建筑数据的添加删除以及存储与查询功能。

基于 HBses 数据库的数据存储技术与传统的基于文件、基于关系数据库和基于面向对象的数据库的存储相比,有着很大的优势:1)Hbase 数据库能存储海量的数据;2)分布式的存储与计算模式将大大提高运行计算速度;3)具有良好的扩展性能,去掉了关系数据库的关系特性,数据之间是弱关系性,非常容易扩展;4)数据模型灵活,无须事先为要存储的数据建立字段,随时可以存储自定义的数据格式。

但本文仅仅是实现了基于 IFC 标准的建筑数据的存储技术。在本文研究的存储技术基础之上还有多项技术有待进一步发展:

(1)如何将非结构化的建筑信息如工程施工合同、工程现场照片、工程变更等文件挂接到 IFC 模型中,进而存储在 Hbase 数据库;

(2)在 Hbase 数据库上再研究开发图形显示界面与编辑平台,使建筑各个参与方可以在该数据库基础上进行前端的编辑操作,进一步完成建筑信息在大数据平台的共享与交互;

(3)在基于 HBase 存储的基础上进一步研究基于 MapReduce 的建筑行业大数据的分析功能,将建筑业与大数据技术真正结合起来。

参考文献

[1] Liebich T. ,Wix J. IFC Technical Guide[EB/OL]2000. http://www. buildingsmart-tech. org.

[2] 李丽娜. 基于 BIM 的建设项目文本信息集成管理研究[D]. 大连理工大学, 2015.

[3] 姜韶华, 李丽娜,戴利人. 基于 BIM 的项目文本信息集成方法研究[J]. 工程管理学报. 2015, 29(4): 101-106.

[4] 董骁. BIM 技术在空间结构中的应用与开发[D]. 上海交通大学, 2015.

[5] 刘照球, 李云贵,吕西林,等. 建筑结构信息集成的程序实现[J]. 沈阳建筑大学学报(自然科学版). 2009, 25(3): 467-473.

[6] Ma L. ,Sacks R. A cloud based BIM platform for information collaboration[C]. International Symposium on Automation and Robotics in Construction, 2016.

[7] 张弛. hbase 数据库访问接口的设计与实现[D]. 北京大学, 2013.

[8] 陈潇睿. 5D 模型在建筑管理软件中的设计与实现[D]. 上海交通大学, 2013.

[9] 孙志佳. 基于 Hadoop 的在线购物原型系统的设计与实现[D]. 东北大学, 2010.

[10] 林良帆. BIM 数据存储与集成管理研究[D]. 上海交通大学, 2013.

[11] Ma Z. ,Wei Z. ,Song W. Application and extension of the IFC standard in construction cost estimating for tendering in China[J]. Automation in Construction. 2011, 20(2): 196-204.

Research on HBase-based BIM Model Storage Technology

Chen Yuan, Yue Shihua

(*School of Civil Engineering*, *Zhengzhou University*, *Zhengzhou* 450001, *China*)

Abstract: With the rapid development of computer and Internet technology, building information model (BIM) has emerged. As a new concept of building technology, BIM has been the driving force of great changes in the construction industry. At the same time, big data and cloud computing technologies have become hot topics. It will be the development trend in future construction industry to combine big data and cloud computing with BIM technology. This paper presents a research about how to store a BIM model in open source HBase database based on big data platform: designing a IFC-based HBase database through the research and parsing of the internal structure of the open standard IFC of BIM model, and then developing IFC-based general library of HBase JAVA API. Finally, the storage method is explained through using the storage example of HBase JAVA API.

Key Words: BIM; IFC; JAVA API; Data Storage